Simulated and Hybrid Annealing Algorithms: Defined, Explained, and Practiced

Jonah Minkoff

October 2025

1 Introduction

Annealing algorithms are heuristic optimization techniques inspired by the physical process of annealing in metallurgy, in which controlled cooling allows a material to settle into a low-energy crystalline state. In computational contexts, annealing refers to gradually reducing a control parameter—the *temperature*—to lower the probability of accepting higher-energy (worse) configurations as the search progresses.

This document presents and contrasts two related optimization methods:

1. **Simulated Annealing (SA):** a classical stochastic optimization method that probabilistically accepts transitions between configurations.
2. **Hybrid Annealing (HA):** a quantum-inspired variant combining classical sampling with quantum tunneling dynamics for improved exploration of complex energy landscapes.

Each section defines the method, outlines its iterative process, and provides a minimal working example on a constrained lattice protein-folding problem.

2 Simulated Annealing (SA)

2.1 Overview

Simulated Annealing is a probabilistic optimization algorithm designed to approximate the global minimum of an energy function $E(x)$ defined over a discrete configuration space. It balances exploration and exploitation by accepting both energy-decreasing and, with decreasing likelihood, energy-increasing transitions as the temperature cools.

2.2 Algorithm Description

Each configuration $x \in \{0,1\}^{n_{\text{nodes}}^2}$ encodes all lattice sites in a compact grid. At each iteration, one bit is randomly flipped to generate a neighboring state.

```
Algorithm: Classical Simulated Annealing

Step 1. Initialize a random configuration $x$.
Step 2. Set the initial temperature $T = T_0$.
Step 3. While $T > T_{\min}$:
    Step 3.1. For $N_{\text{sweeps}}$ iterations:
        Step 3.1.1. Generate a candidate configuration $x'$ by flipping one randomly chosen bit of $x$.
        Step 3.1.2. Compute the energy change $\Delta E = E(x') - E(x)$.
        Step 3.1.3. If $\Delta E \leq 0$, accept $x \leftarrow x'$; otherwise accept with probability $p = e^{-\Delta E/T}$.
    Step 3.2. Update temperature $T_{k+1}$ according to the chosen cooling schedule (linear or geometric).
Step 4. Return the final configuration $x$.
```

2.3 Implementation Notes

- Each update corresponds to flipping a single, randomly chosen bit.
- One *sweep* consists, on average, of one attempted update per bit variable (i.e., n_{bits} total flips).
- Transitions are accepted according to the Metropolis rule, based on the Boltzmann factor $e^{-\Delta E/T}$.
- In [1], the cooling schedule is geometric:

$$\beta_i = 1.05^i, \quad (i = 1, \dots, 25)$$

  corresponding to 25 temperature steps.

2.4 Parameter Definitions

- x: Current configuration (bit string of length n_{nodes}^2).
- $E(x)$: Energy (or cost) of configuration x.
- $T_0, T_{\min}$: Initial and minimum temperatures.
- T_k: Temperature at iteration k.
- N_T: Total number of temperature steps (typically 25).
- N_{sweeps}: Number of sweeps per temperature; each sweep $\approx$ one update per bit

- α: Cooling factor. For linear incremental cooling,

$$T_{k+1} = T_0 - \frac{k}{N_T}(T_0 - T_{\min})$$

- Typical empirical runtime: 1.1 ms per sweep (on an Intel i9-13900K, as in [1]).

2.5 Paper-Specific Implementation

For SA computations:

- All runs spanned the same set of 25 geometrically distributed temperatures:

$$\beta_i = 1.05^i, \quad i = 1, \ldots, 25$$

- Updates were single-bit flips, controlled by a Metropolis acceptance criterion.
- To assess runtime dependence, runs comprised between 1000 and 80,000 sweeps per temperature, where one sweep corresponds on average to one attempted update per bit variable.
- One sweep required approximately 1.1 ms on an Intel Core i9-13900K processor.
- For each sequence and run length, a set of 100 runs was performed using different random number seeds.

2.6 Cooling Dynamics

As the temperature decreases, the acceptance probability for energy-increasing moves falls exponentially. Early in the process, this allows the system to escape local minima; later, the cooling schedule biases the search toward convergence. The linear schedule offers predictable convergence, while geometric cooling better preserves exploration depth.

2.7 Discussion

Simulated Annealing efficiently explores discrete configuration spaces by allowing occasional uphill moves early in the process, preventing premature convergence. As temperature decreases, transitions become increasingly selective, guiding the system toward low-energy configurations. This makes SA well-suited to problems with rugged energy landscapes, such as lattice protein folding. Its bit-string representation and per-bit updates align neatly with discrete lattice constraints.

The next section extends this foundation into a *Hybrid Annealing* model, which incorporates quantum-inspired dynamics and enables parallel exploration across coupled configurations.

3 Hybrid Annealing (HA)

3.1 Overview

Hybrid Annealing is a quantum-classical optimization method designed for large, discrete problems formulated as Quadratic Unconstrained Binary Optimization (QUBO) problems. It combines classical decomposition and iterative solution updates with quantum annealing (QA) subroutine calls.

The classical routines maintain feasibility and integrate partial solutions, while quantum annealing exploits tunneling dynamics to explore subspaces of the energy landscape that may be difficult for purely classical approaches. This hybrid approach enables the system to more efficiently approach low-energy configurations in rugged optimization landscapes.

3.2 Algorithm Description

Given a QUBO problem $Q(x)$ over n binary variables, the goal is to minimize the energy function:

$$E(x) = x^\top Qx$$

Algorithm: Hybrid Annealing

```
Step 1. Initialize a classical state $x_{\text{classical}} \leftarrow$ random feasible solution.
Step 2. Set the initial quantum anneal runtime $t_{\text{anneal}} \leftarrow 6$ s (empirical default).
Step 3. Set the success flag: success $\leftarrow$ False.
Step 4. Repeat until success or maximum allowed runtime:
  Step 4.1. Perform a fixed number of QA runs (e.g., 100):
    Step 4.1.1. Classical decomposition: extract a subproblem from $x_{\text{classical}}$ focusing on
                the variables contributing most to high energy:
                    $\text{subproblem} \leftarrow \text{extract\_subproblem}(x_{\text{classical}}, Q)$
    Step 4.1.2. Quantum annealing: solve the subproblem using the QPU for the given
                runtime:
                    $\text{quantum\_result} \leftarrow \text{QPU\_anneal}(\text{subproblem}, t_{\text{anneal}})$
    Step 4.1.3. Decoding: convert the quantum output into a classical solution for the
                subproblem:
                    $\text{sub\_solution} \leftarrow \text{decode}(\text{quantum\_result})$
    Step 4.1.4. Classical integration: update the full classical state with the sub-solution:
                    $x_{\text{classical}} \leftarrow \text{update\_solution}(x_{\text{classical}}, \text{sub\_solution})$
```

Step 4.1.5. Compute the current energy:

$$E(x_{\text{classical}})$$

Step 4.1.6. **Check for success:** if $x_{\text{classical}}$ meets solution criteria, set success $\leftarrow$ True and break the loop.

Step 4.2. **Runtime tuning:** if no solution found, increment the anneal time:

$$t_{\text{anneal}} \leftarrow t_{\text{anneal}} + 2 \text{ s}$$

Step 5. Return $x_{\text{classical}}$ and the total runtime used.

3.3 Parameter Definitions

- $x_{\text{classical}}$: Current classical configuration of all binary variables.
- Q: QUBO matrix defining the optimization problem.
- $E(x)$: Energy (objective) function for a given configuration.
- subproblem: Subset of variables extracted for QA to focus on the most energetically significant bits.
- t_{anneal}: Runtime allocated to each quantum annealing query.
- success: Boolean flag indicating whether a solution has been found.
- quantum_result: Raw output from the QPU for the subproblem.
- sub_solution: Decoded, feasible classical solution corresponding to the quantum output.

3.4 Paper-Specific Implementation

For HA computations:

- The hybrid approach enables tackling problems with many thousands of fully connected variables, far beyond what is feasible using only the QPU.
- Unless specified, the D-Wave hybrid annealer selects the runtime automatically based on problem size; this default value is also the shortest allowable runtime.
- HA computations were performed for one or more run-times per sequence. For each runtime, a set of 100 runs was generated.
- Starting with the default runtime (6 s), the runtime was increased in steps of 2 s until all 100 runs returned the correct solution, with maximum required runtimes varying between 6 and 22 s across the six sequences.

3.5 Notes on Implementation

- The classical decomposition selects a tractable subset of variables suitable for the QPU, typically the ones with highest contribution to the current energy.
- Only the variables in the subproblem are updated by QA; the rest of the classical state remains unchanged.
- Multiple QA runs per iteration improve the chance of finding low-energy configurations.
- Runtime tuning is empirical, gradually increasing anneal time for harder subproblems.
- This hybrid loop allows iterative refinement: the classical state evolves using quantum-assisted partial updates until a sufficiently low-energy solution is found.

References

[1] Anders Irbäck, Lucas Knuthson, Sandipan Mohanty, *Folding Lattice Proteins Confined on Minimal Grids Using a Quantum-Inspired Encoding*, arXiv:2510.01890, 2025. `https://arxiv.org/abs/2510.01890`

[2] Georgia Institute of Technology, OMSCS 7641 Course Notes, *Simulated Annealing: Methods and Real-World Applications*, 2024. `https://sites.gatech.edu/omscs7641/2024/02/19/simulated-annealing-methods-and-real-world-applications/`